Andalucía lab CENTRO DE INNOVACIÓN TURÍSTICA

Junta de Andalucía
Consejería de Turismo, Regeneración, Justicia y Administración Local

UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional

PROGRAMA DE ATRACCIÓN DE TALENTO TECNOLÓGICO PARA EL SECTOR TURÍSTICO ANDALUZ

2020

INTELIGENCIA ARTIFICIAL. 1ª PARTE

FORMACIÓN TURISMO Y TECNOLOGÍA

UTE conformada por:

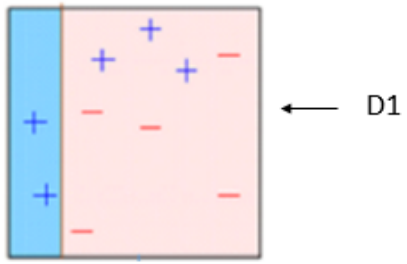
ideanto comunicación y tecnología

arelance Building Digital Talent

Boost

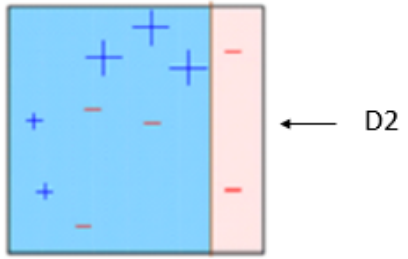
Boost

- Una de las técnicas utilizadas para mejora la exactitud de los árboles de decisiones es el Bagging, el cual consiste en coger subconjuntos de elementos, generar árboles con ese subconjunto y luego unirlos en un único “gran árbol”. Esta es la técnica usada por Random Forest
- Existe otra técnica llamada boosting, la cual intenta encontrar el mejor ajuste posible para incluir la mayor cantidad de datos incorrectamente clasificados. Para ello en una primera iteración, realiza un árbol de clasificación y observa qué datos han sido incorrectamente clasificados

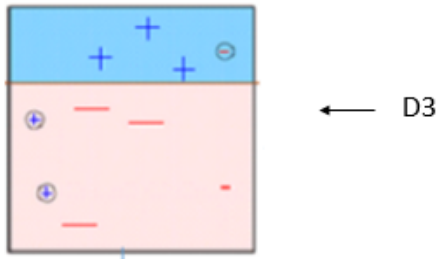


Boost

- A partir de ahí vuelve a ejecutar de nuevo el árbol dando mucho mayor peso a los datos incorrectamente clasificados (en este caso las 3 cruces superiores)



- Este aumento del peso hace que otros elementos pasen a estar incorrectamente clasificados (en este caso las 3 rayas rojas inferiores). Es por tanto que vuelve a dar un peso mayor a esos elementos y vuelve a hacer un árbol de clasificación



Boost

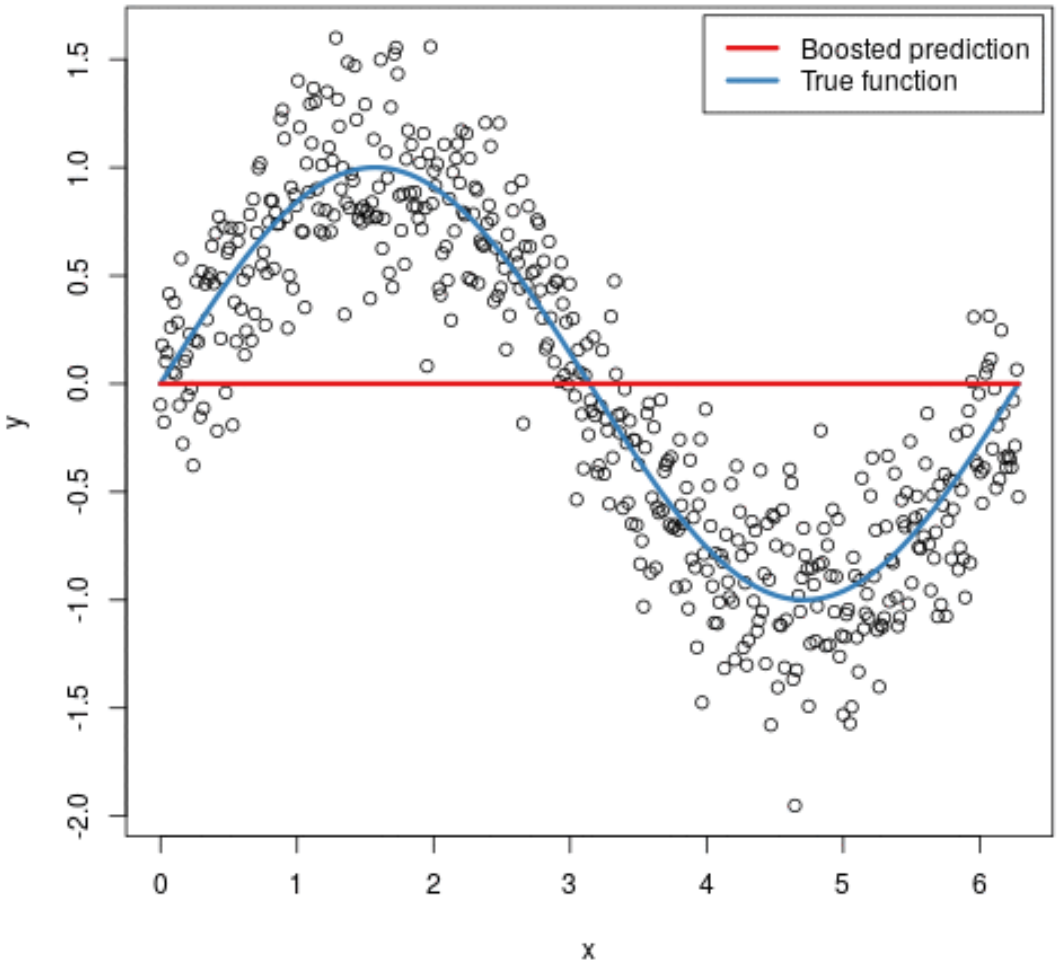
- Finalmente va combinando y encontrando las reglas en base a intentar dejar el mínimo numero de elementos incorrectamente clasificados

← D4

- Este algoritmo es iterativo, por lo que va generando una y otra vez el árbol intentando dejar el mínimo número de elementos incorrectamente clasificados.
- En un límite teórico, dejándolo correr de forma ilimitada, tendríamos que el algoritmo para a clasificar correctamente todos los elementos del conjunto. No obstante eso provocaría un modelo fuertemente sobreajustado. Es por ello que debemos dejar que el algoritmo ejecute iteraciones hasta justo el momento en el que comencemos a detectar el sobreajuste.
- En C5.0 podemos incrementar el número de iteraciones para boosting mediante el parámetro trials=n (por defecto trials=1)

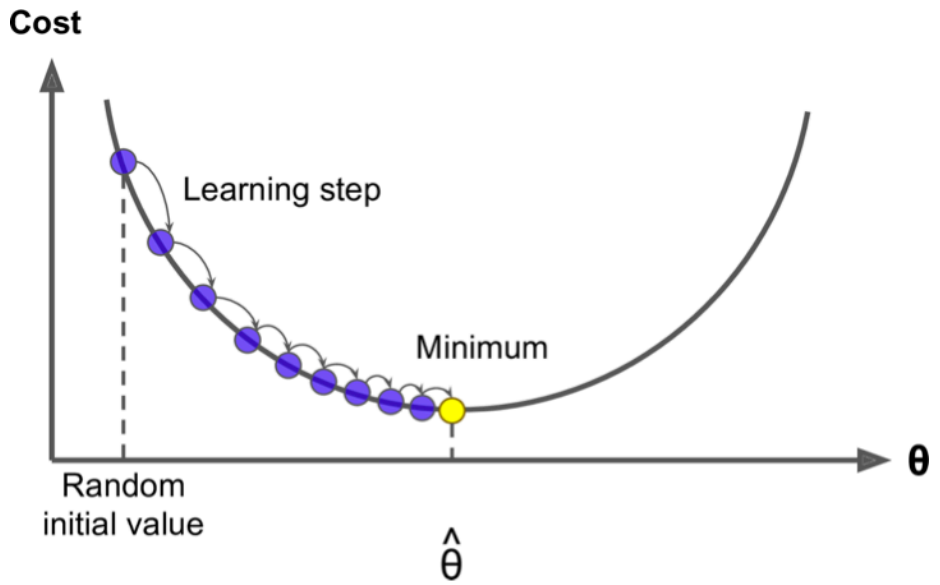
Boost

- Uno de los algoritmos más populares que utiliza Boosting es el Gradient Boosting, el cual trata o bien de encontrar la mínima cantidad de errores en un clasificador o bien en realizar un descenso del gradiente, minimizando una función de coste.
- Con él se construye un árbol básico de toma de desiciones y va iterándose sucesivamente hasta ir logrando mejorar la precisión
- Las diferencias con Random Forest son:
  - Puede trabajar con NAs
  - Random Forest necesita que las variables a usar estén balanceadas y que haya aproximadamente el mismo número de casos en cada una de ellas. Gradient Boosting puede hacer precisiones en caso que una de las variables esté desbalanceada y haya muy pocos casos respecto al resto
- Pero…
  - Sobrereacciona e intenta ajustar lo máximo posible al modelo, por lo que el peligro de overfitting es extremo
  - Es computacionalmente muy costoso
  - Necesita un ajuste fino de parámetros que hay que considerar
  - Es poco interpretable



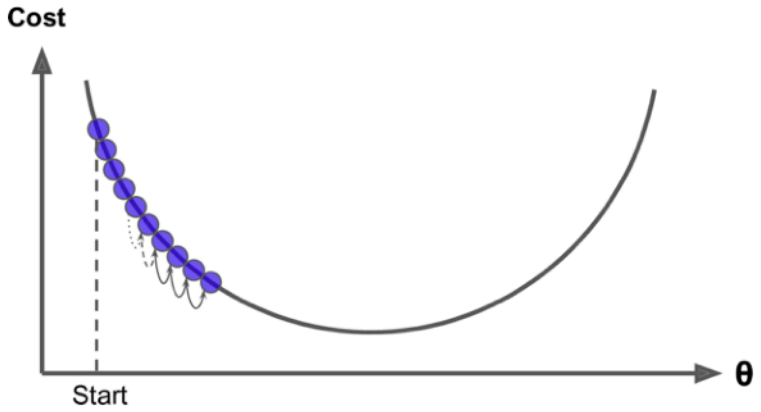
Boost

- Descenso del gradiente
  - Este tipo de algoritmo intenta minimizar una característica (desviación cuadrática media, índice de gini, …) a través de descensos del gradiente

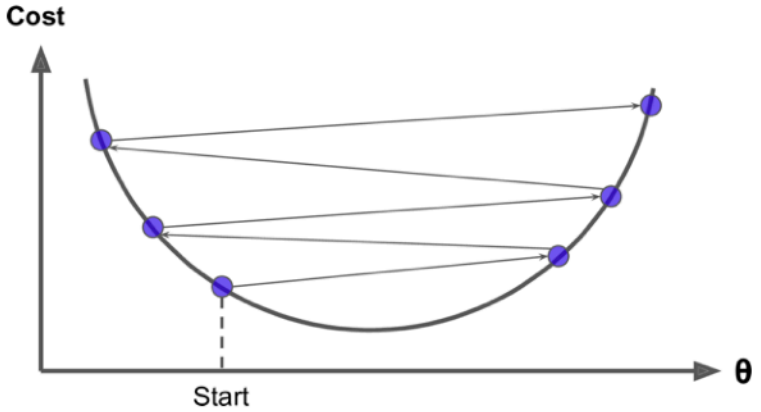


Boost

- Descenso del gradiente
  - Un parámetro importante en el descenso del gradiente es la “tasa de aprendizaje”, la cual indica el tamaño del paso en el descenso.
  - Si el paso es muy pequeño, tardará mucho tiempo en converger a la solución
  - Si el paso es muy grande, puede que no llegue a converger a la solución



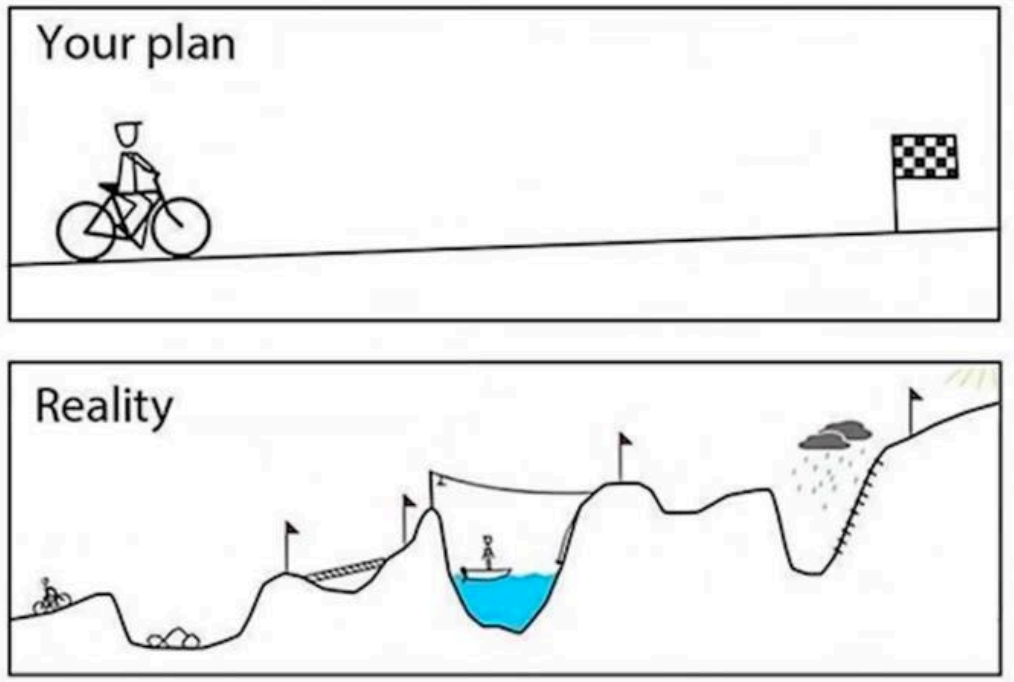
a) too small



a) too big

Boost

- Descenso del gradiente
  - Otro problema con el descenso del gradiente es que la función tenga mínimos locales que impidan dar con el mínimo global (la solución)



Boost

- Parámetros a considerar en un Gradient Boosting (gbm):
  - Número de árboles (n.trees): Si proporcionamos un número muy grande de árboles, el algoritmo provocará un overfitting intentando ajustarse lo máximo posible a los datos. Debemos de elegir un número adecuado para evitar este sobreajuste
  - Profundidad de los árboles (interaction.depth): Cada vez que tiene ramificar una decisión este parámetro indica el número de ramas que va a generar. Un parámetro que suele funcionar bien en la mayor parte de los casos es 1, que indica que va a generar una nueva rama, no obstante hay problemas que pueden requerir un parámetro de ramas mayor
  - Tasa de aprendizaje (shrinkage): Controla la velocidad con la que "descenderemos por el gradiente" (es decir, probaremos soluciones más profundamente "agresivas" intentando localizar la solución. Un parámetro pequeño indicará que el algoritmo irá a pequeños pasos en la búsqueda de la solución, haciendo que pueda llegar a encontrar una solución menos sobreajustada que si lo forzamos a buscar una convergencia muy rápida
  - Submuestreo (bag.fraction): Indica si vamos a usar el 100% de las muestras o tomaremos un subconjunto de datos. En ese último caso lograremos reducir el sobreajuste del modelo
  - Distribución (distribution): Qué tipo de función de coste vamos a usar:
    - gaussian: Equivale a una función de coste cuadrática. Por defecto cuando se usa valores numéricos a predecir
    - bernouilli: Equivale a una función regresión logística entre 0 y 1. Por defecto cuando la salida es una variable de factor con 2 valores posible
    - multinomial: Generalización de una clasificación de una variable de factor
  - Intentos de validación cruzada (cv.folds): Presenta el número de validaciones cruzadas (80/20) que va a realizar antes de dar por bueno un árbol

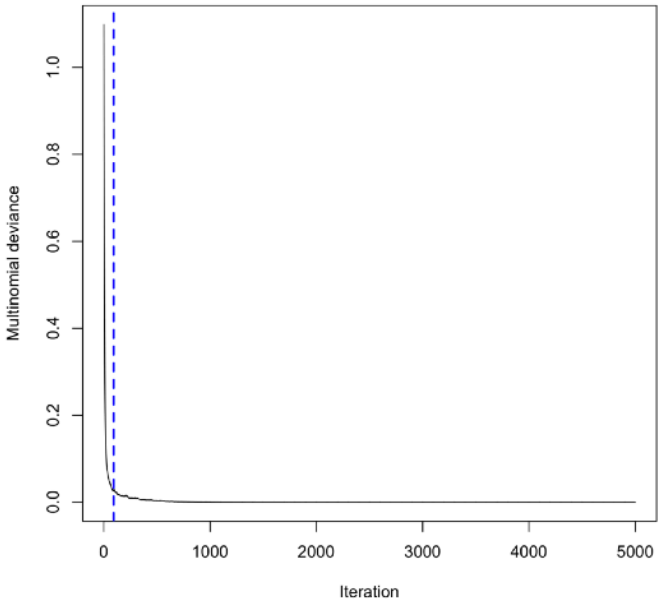
Boost

- Gradient Boosting de iris (multiclasificador):
  - model = gbm(Species~., data=iris, n.trees=5000)
- Obtención de los árboles óptimos
  - gbm.perf(model)
- Lanza un warning indicando que no tiene claro resultados y que necesitaría una cross-validation
  - model = gbm(Species~., data=iris, n.trees=5000, cv.fold=5)
- Predicción
- Si cv.fold ha sido especificado, no hay que indicar el número de árboles
  - r = predict(model, newdata=iris, type="response")
- Si cv.fold no se ha indicado, hay que indicar el número de árboles óptimo
  - r = predict(model, newdata=iris, type="response", n.trees=95)
- r contiene la probabilidad de ser cada una de la especie
  - r
- Si queremos averiguar cuál es la indicada, hay que preguntar por el valor máximo de cada fila
  - rCat = apply(r, 1, which.max)

Boost

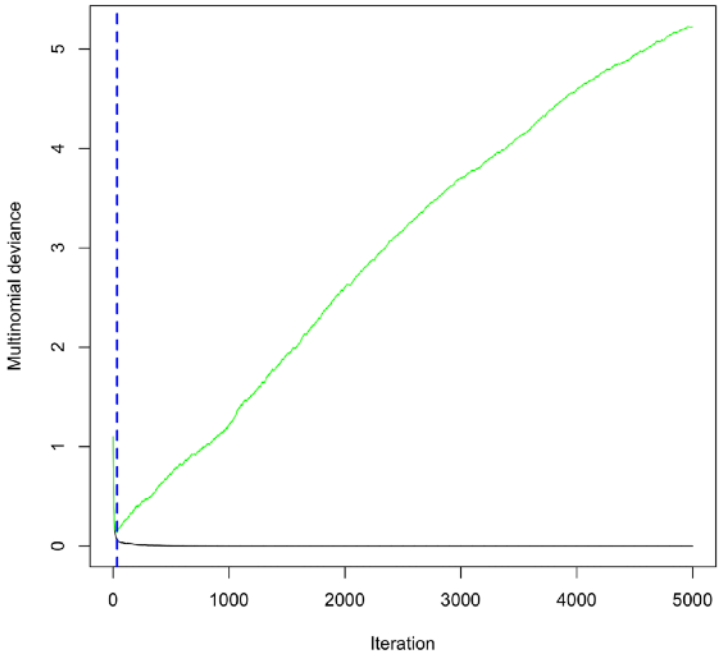
- Gradient Boosting de iris (multiclasificador):
  - model = gbm(Species~., data=iris, n.trees=5000)
- Obtención de los árboles óptimos
  - gbm.perf(model)

- A cada iteración que haga, encontrará una mejoría en el modelo, obteniendo la la suma de las desviaciones total.
- Llega un momento en el que a cada iteración que se haga, no se consigue más que una mejora asintótica del modelo. ¡En ese "codo" es el punto en el que podemos comenzar a considerar que el modelo está comenzando a sobreajustar y por tanto ese es el número de iteraciones óptimo!
- No obstante en este caso, da un warning diciendo que necesita una cross-validation porque no se "fía" de hacer una única división train/test (el modelo detecta que hay muy pocos datos)



Boost

- Así pues, indiquémosle que queremos una cross-validation de 5 divisiones:
  - model = gbm(Species~., data=iris, n.trees=5000, cv.fold=5)
- En teoría habría que indicar que la distribución es multinomial (de factores), pero es la que asume por defecto si no decimos nada
  - model = gbm(Species~., data=iris, n.trees=5000, distribution="multinomial")
- Con eso nos aseguramos que la mejora en la iteración es real y no causada por el azar de la elección del train/test a azar
  - gbm.perf(model)
  - [35]
- Esto nos dice que 35 iteraciones es el valor óptimo para este modelo
- Para la predicción
  - r = predict(model, newdata=iris, type="response")
- Para averiguar qué tipo de especie es, hay que averiguar el valor máximo por cada fila
  - rCat = apply(r, 1, which.max)



Boost

- Vamos con el Bank Marketing
  - model = gbm(y~., data=bank, n.trees=5000, cv.fold=5)
- ¿Por qué sale el error? Porque asume una distribución binaria y actúa para intentar predecir el valor entre 0 y 1
- Hay que indicarle que es una distribución multinomial
  - model = gbm(y~., data=bank, n.trees=5000, cv.fold=5, distribution="multinomial")

Boost

- Vamos con el Consumo de Agua de Sevilla. Dos cosas: recuerda el complete.cases() y también quita las variables que no quieras usar (no funciona bien el + para agregar variables):
  - model = gbm(CONSUMO ~ ., data=aguaSINNA, n.trees=5000, cv.folds=5, distribution="gaussian")
- Aquí la distribución es gaussiana